PRESERVING PROGRAM SECURITY IN DISK BASED

MICROCOMPUTER SYSTEMS

by

Henry A. Roberts, Jr.

Bachelor of Arts

Erskine College, 1970

Submitted in Partial Fulfillment of the

Requirements for the Degree of Master of Science

in the Department of Computer Science

University of South Carolina

1981

Department of Computer Science
Director of Thesis

Robert Trueblood
Department of Computer Science
Second Reader

George M. Reeves
Dean of the Graduate School

ACKNOWLEDGEMENTS

I would like to express my deepest appreciation to Professor Deogun and Professor Robert P. Trueblood for their constant encouragement, concern and support. Without them this thesis could never have been finished. I would also like to express my gratitude to Dr. George Johnson, then Graduate Director of the combined Mathematics, Computer Science, and Statistics Dept., who gave me my chance and encouraged me to begin work on this degree. Finally, I would like to express my appreciation for my wife who stood by me and encouraged me even unto the early morning hours, when all appeared hopeless.

TABLE OF CONTENTS

CHAPTER I

BACKGROUND ON MICROCOMPUTERS

This chapter serves as a background concerning the evolution and development of the APPLE computer. In addition the APPLE'S place among computers today is discussed.

The origin of any computer must of course have as its beginning the Princeton machine and the genre of mainframes which followed. As technology increased new methods were found to store data and create the large number of logic gates required to implement a Central Processor Unit (CPU).

The discovery of the transistor led to an evolution in computers. Digital Equipment Corporation developed and marketed the PDP 12 in 1968. Although physically large by today's standards, it was the first of a new breed of mini-computer. It's logic gates were created from transistors and were housed in two six foot equipment racks.

A 12 bit machine, it organized its memory in 256 byte pages and began a trend toward smaller memory efficient operating systems, compilers, and user programs. The mini computer stabilized into a 16 bit machine with aproximately 64 thousand bytes of memory.

Other companies quickly followed with their own versions of the mini-computer, most notable was DATA GENERAL with their NOVA series. The trend was an accumulator based CPU with a stack register and status register based architecture. Technological developments in the integrated circuit led to the development of the LSI chip or Large Scale

Integration. This produced a silicon chip with the equivalent of hundreds to thousands of integrated circuits in an area less than a sixteenth inch square. CORE memory began being replaced by semi-conductor memory. Integrated circuit technology increased rapidly. The mini-computer shrank to a few circuit boards on a back plane. The back plane was nothing more than a circuit board itself with connections to a series of circuit board connectors. These connectors both held a special purpose circuit board, and provided the electrical connections to the back plane and other circuit boards. The mini-computer had shrunk to the size of the proverbial bread box and with it its memory. Such a mini computer could easily house sixteen to thirty two thousand bytes of RAM memory. To define terms, a micró-computer is a computer which uses a single LSI ingegrated circuit for all of its central processor functions. These single integrated circuits have numbered designations, such as 4004,6502, 8080 etc.

The mini-computer began to evolve, itself, into the micro-computer. The same LSI revolution which lead to the drastic reduction in the size of the mini-computer developed a simple computer on a single silicon chip. The process was gradual. By the Early 70s a PDP 12 E CPU consisted of several LSI chips on a single circuit board. The first of the single chip CPUs was the 4004 by INTEL. This was a four bit machine with a primitive instruction set and limited memory addressability. Never the less it constituted a

break through in computer technology. The mini-computer was largely used in a few research establishments and as a controller for industrial machinery. The INTEL 4004 opened the way to a small, inexpensive single board replacement for a relatively expensive mini-computer. Development of the micro-computer chip was rapid. An eight bit version was developed by INTEL with the 8008 part number. It was essentially an eight bit 4004. This led to greater efficiency and simpler programming. The hobby market began to take notice. Articles began to appear in magazines such as POPULAR ELECTRONICS. These articles described micro-computers as the wave of the future. Serious computer users scoffed the idea. An article in Popular Electronics described a small general purpose micro-computer built around the 8008. The design was that of the mini-computer. It used several circuit cards and a back plane. One card contained the CPU, others were dedicated to memory functions and still others handled I/O. Boot strapping was accomplished by the use of a front panel with switches. These would load a byte of information at a time into a memory location. It is significant to note that although the accumulator and data busses were eight bits wide, the address busses followed the mini-computer standard of sixteen bits. Thus, the micros were able to address aproximatelly

The landmark development occured with the development of the INTEL 4040 and 8080. The latter with its enhanced instruction set and power became the first micro-computer

chip to find its way in a computer kit which imulated the mini-computer. Altair developed an 8080 based micro-computer which closely resembled the mini-computers from DEC and DATA GENERAL. It was programmable from switches in the front pannel and perform general purpose computing. Interfaces connected it to teletypes and operating systems allowed machine language programming. Gradually, assembly language and BASIC became available.

Other small corporations came into being. All developed 8080 based processors around Altair's S-100 buss. A buss is a series of connections from one electrical device to another. An entire magazine devoted to these new micro-computers came into being. BYTE magazine was published by amateur radio operator Wayne Green, who at the time also published a popular amateur radio magazine.

Another development was MOTOROLA's 6800 processor chip. It represented major improvements in conditional branching, and indexing. ALTAIR developed a processor around it, but it remained for South West Technical Products to develop an effective micro-computer. South West Technical Products utilized MOTOROLA's boot strap ROM and operating system. South West Technical Products' computer was still back plane based and followed the physical configuration of the mini-computers. The use of MOTOROLA's ROM, however, led to a standardization which produced several independent software houses. This was possible because of the standard I/O and cassette loading techniques which were located on the boot

strap ROM.

The really big breakthrough, however, was underway in a garage in southern California. Steve Wozniack and Steve Jobs developed a computer on a single circuit board. [ARKPH] The purchaser needed only to build a power supply, connect a keyboard, and provide a cassette tape recorder and video monitor. They called it an APPLE computer. It contained space for 48K of dynamic RAM, a cassette interface, video generator and a powerful monitor ROM together with Integer BASIC in ROM. This BASIC interpreter executed a series of bench marks much faster than other processors including a LSI 11. It produced color praphics with a few simple BASIC commands, and a speaker took the place of the old teletype bell. The monitor contained a disassembler as well as routines for block memory moves, comparing two segments of memory, and color graphics. The Integer BASIC ROMs also contained a mini-assembler, a 16 bit computer emulator, and floating point subroutines. The entire package sold for $400. This quickly began to take over the hobby market. Other companies such as Radio Shack and Ohio Scientific took notice and began developing their own products.

APPLE computer completely packaged their device. The purchaser needed only a cassette tape recorder and video monitor. The business world began taking notice and APPLE computer bought the rights to a fast floating point BASIC. This BASIC executed the same series of benchmarks in just twice the time of the DEC LSI 11 version of the PDP 11. As

popularity increased, magazines began publishing regular features on the APPLE computer and software houses came into being which served this special segment of the market.

CHAPTER II

THE APPLE ARCHITECTURE

The Mostek 6502 processor is a refinement of the 6800 by MOTOROLA. It contains six registers which are appearant to the user:

1. The program counter--unavailable to the user
2. The status register--can be read by user
3. The stack pointer--which is fully accessable
4. The accumulator
5. The X register--used for indexing
6. The Y register--used for indexing

The program counter is a sixteen bit register which contains the address of the next instruction to be executed. It is completely unaccessable to the user. Its contents may not be read, set or altered. The only technique available for changing the program counter to a number of different locations is to push the new address on the stack and execute a return from subroutine.

The status register is an eight bit register which contains the following:

1. Zero flag: This is set if the last referenced register or memory location is set to zero.

2. Negative flag: This is set if the left most bit of the last referenced register or memory location is set to one.

3. Overflow flag: This is set if the carry bit is one.

The stack pointer is an eight bit register which is incremented or decremented by the use of PUSH, PULL, JSR, or RTS instructions. The first two instructions push information onto the 6502 stack, while JSR jumps to a subroutine and RTS returns from subroutine. The stack pointer's designed purpose is to save the return address for the subroutine being called. It functions as a pointer to the return addresses in page one. Internally an additional bit occupies the position left of the most significant eighth bit. This gives it an effective range in hex of $100 to $1FF. Upon reset, the stack is set to $FF and each jump to subroutine decrements the value by two (the '$' in front of a number is microprocessor notation for hex). A PUSH instruction decrements the value by one and a POP increments it by one. As the stack pointer is decremented, a byte is stored at its previous location. Thus the stack pointer points to the next available byte in memory. In the case of the JSR instruction, the program counter is stored in page one; least significant byte first. PUSH and POP transfer the contents of the accumulator to the stack.

The Accumulator is an eight bit register plus an additional carry bit. It supports add with carry, subtract with carry, load and store operations, left and right shift, and transfers to and from the X and Y registers.

The X register is an eight bit index register. its capabilities are broader than the eight bit Y register. The

difference is described below in the next paragraph on indexing. Both index registers may be incremented, decremented, loaded or stored in memory. The 6502 supports both load immediate instructions and load from memory. As mentioned above, both registers may be transfered to and from the accumulator.

The 6502 handles indexing in several fashions. Both direct and indirect indexing is supported. The format is the command followed by a one byte page zero address or a two byte address. The addresses, however, can point to several locations, depending on usage. A page zero address may contain the base address of the table being indexed. An example of the assembler command to do this is:

```
LDA (26),Y
```

which loads the accumulator with the byte located at the address pointed to by locations $26 and $27 offset by the value of the Y index register. A similar command uses location twenty six as a base address, indexes it by the value of X and the resultant byte along with the one following it is the desired address. This command is only possible with the X register. An example of this command is of the form:

```
LDA (26,X)
```

The 6502 supports eight conditional branch instructions. Branch instructions allow a bi-directional offset and are composed of two bytes. The first byte is the instruction, and the second byte is the two's complement value of the offset. Conditional branches are allowed on carry

clear and carry set, equal to zero and not equal to zero, plus and minus, and finally on overflow set and clear conditions. One branch label is misleading. The branch on plus command is in reality a branch on not minus, therefore zero is is assumed to be plus.

A final subject in 6502 architecture is its handling of interrupts. There are three classes of interrupts, maskable, nonmaskable, and reset. The 6502 handles all three classes of interrupts by branching to one of three locations in the top six bytes of memory.

APPLE II

The architecture of the APPLE II microcomputer is 6502 based. This discussion below covers memory usage, both APPLE II monitors, page zero usage, both Integer and floating point BASICS in ROM and DOS.

The memory of the APPLE II is divided into two hundred and fifty six byte pages. Page zero contains addresses and information needed by the system. This allows DOS, BASIC, and the monitor access to the same information.

The original APPLE contained as little as four thousand bytes of memory. Today an APPLE computer may have sixteen, thirty two, or forty eight thousand bytes of memory. The importance of page zero in program security lies in the fact that several locations are changed upon reset. The contents of most of these locations cannot be changed with out adversely affecting the overall computer operation, or the locations themselves are modified by the monitor and thus any

information stored there would be lost almost immediately. There are, however, three locations which can be modified under the right conditions. These locations are $31, $32, and $48. Location $32 controls the inverse video mode(i.e. black characters on a white background) and thus is limited to games which display graphics only. One exception to this rule would be to use location $32 as a temporary storage area, replacing the original contents before any video is printed on the screen. Location $48 is currently used by DOS and it is usable only if DOS itself is not present or is modified in such a manner as to utilize a diffeent page zero location. The final location is $31. Although this location is changed by operations from the monitor mode, none of the hight level languages utilize it, and thus it is usable under most conditions.

Page two is used as a keyboard buffer by the monitor and both versions of basic. The keyboard input routines store the ASCII in page two for each key pressed here until a RETURN. This makes it ideally suited as a location to store especially important routines. It is not difficult to provide alternate keyboard input routines and any data or routines stored at page two must as a necessity be destroyed by attempts to operate the computer other than from the executing program.

Page three is used in the booting process of DOS versions 3.1-3.21. DOS 3.3 ignores it entirely, as does the rest of the system. After the DOS has been booted, this page

of memory is not used further. As a result, many BASIC programmers locate their short machine language routines here.

Pages four through seven are the screen buffer. The majority of these locations are destroyed by any attempt to RESET the computer. Since there is a second area of memory which can serve as a screen buffer this has been used as a location for sensitive routines in a number of machine language programs which either use graphics entirely, or the second screen buffer. The main disadvantage involved with using the second screen buffer is that the user must develop his own print routines, as the monitor is limited to only printing to the first screen buffer.

Pages eight through eleven ($800-BFF) comprise the second screen buffer. $800 is also the beginning location for APPLESOFT programs, Integer variable space, and most unprotected machine language programs.

The remaining pages of memory may be used for APPLESOFT variable storage, DOS, or user defined functions. In a forty eight thousand byte environment, DOS begins at $9D00 and continues to the end of memory at $BFFF. DOS is discussed in further detail in the section 'APPLE Disk Operating System'.

APPLE has two versions of BASIC. The earliest version was Integer BASIC. Besides being limited to sixteen bit integers, INTEGER BASIC lacks string handling functions as well as transendental functions and has no provisions for error handling. The latter is significant since it allows

the program to be stopped with a control C. All early programs were written in integer basic. The majority of business programs, however, are written in APPLESOFT. At the present time, APPLESOFT has begun to be used with many types of games and a significant number are entirely machine language.

APPLE's monitor is a basic utility tool. It has routines for block memory moves, block comparisons of memory, a disassembler (INTEGER BASIC contains a mini assembler), routines to handle all screen functions, low resolution graphics routines, as well as memory examine, memory change, keyboard input, printing, and mass storage via cassette tape recorder.

From the standpoint of copy protecting software, the monitor makes the task difficult. The ability to block move, disassemble, assemble new routines as well as examine memory locations and change them gives a great deal of machine level power to the user. In addition, the standard monitor ROM which comes with INTEGER BASIC drops to the monitor level as a result of pressing the RESET key. The current model monitor ROM is termed the AUTOSTART ROM. It derives its name from the fact that upon initial start up it attempts to load DOS into memory and execute it. This allows for turn-key systems. From a protection stand point, a more important feature is the programmer's ability to define the reset function. It can be trapped as an error, restart the program, or whatever other function the programmer

desires.

APPLE'S DISK OPERATING SYSTEM

The final subject of discussion regarding APPLE archatecture concerns DOS itself. APPLE DOS consists of two major parts, the read/write subroutines (RWTS) and the executive routines with include the I/O routines, parser, and logic required to translate commands into reading and writing single two hundred and fifty six byte blocks of data. Thus the sequence of events from a load command to the program's actually being loaded is as follows:

1. The LOAD command is parsed and control passed to the load module.

2. The catalog is searched for the name of the required file.

3. The list of track and sector numbers relating to the file is loaded.

4. Successively, each file is loaded into memory a block at a time by RWTS.

5. Upon loading the last file, control is returned to the I/O routines.

DOS has provision for both types of BASIC files, binary files, and text files. It automatically selects the appropriate type of BASIC if required and signals an error if it is not present. Upon system start up, DOS loads and runs what is known as the greeting program. This of course al-

lows for a turn key system. The greeting program is named at the time the disk is initialized. Initialization formats the disk into tracks and sectors, saves a copy of the DOS itself (which must load into the same area of memory as the originatting DOS) and saves the greeting program. Such a disk is termed a slave disk since it cannot relocate itself into any part of memory.

DOS has its own error I/O routines and error checking routines. When DOS is connected to the keyboard buffer via addresses in page zero, it first checks the keyboard buffer for commands or data. The parser attempts to recognize any commands which are intended for it. If part of the command is improperly stated, it prints an appropriate error message. If it cannot find the command in its command table, it passes the command over to BASIC. If BASIC cannot recognize the command it prints an error message. The LOAD and SAVE commands are special since either DOS or BASIC may process them. If the command has an argument, DOS assumes the command is intended for itself. If the command has no argument, the command is passed to BASIC and BASIC's cassette routines handle it.

CHAPTER III

THE SOFTWARE PIRACY PROBLEM

The stand alone microcomputer was marketed in 1977. In two years the popularity of the APPLE and TRS-80 microcomputers had attracted numerous programming efforts. Software houses devoted to one or both machines appeared. Advertising was lavish and wide spread. Beginning in 1978, there was a virtual flood of programs as the combined efforts of the software houses became evident. The game programs were the first efforts, and although the APPLE was originally intended for hobby use, business programs began appearing.

It is important to note a significant difference between these types of programs. Business packages require much in the way of documentation, support, and updates. Game programs, on the other hand, tend to be short term successes. They frequently require little in the way of documentation. They do not require a great deal of updating if designed well in the beginning. Failures are not so obvious, and are not prone to causing a large scale economic loss. This gives the purchaser of the game a great deal of independence. On one hand the business user is dependent on the software house, and on the other hand the game user has no such dependence.

The two classes of purchasers also differ in economic capability. Business systems, by their very nature, are sold to businesses which consider a multi-thousand dollar

purchase an ordinary cost of business. Games, however are most frequently purchased by individuals whether or not they use them at home or on their business computer. The resources are in this case much smaller. If you consider the average price of a game today to be in the twenty-five to forty dollar price bracket, then simple arithimetic yields totals of a thousand dollars and more just to purchase the top thirty games. "Softtalk" magazine (a restricted periodical sent free of charge to APPLE users) lists the top thirty programs each month. Approximately a fourth of these change each month and sometimes greater than a third of the programs drop from popularity only to be replaced. Thus, an individual can spend from two hundred to four hundred dollars each month to keep up with the currently popular games. This author is not suggesting that individuals do in fact attempt to keep up in such a fashion, but heuristics would indicate that there is some degree of this tendency in every APPLE owner.

This leads inescapabily to the conclusion that the avarage individual cannot support such a situation. In fact this is not the case. The fact is that individuals soon learned that it was both simple and good economic sense to share copies of games with friends. Thus, if ten friends owned APPLE computers, each person could keep up with the popular games for twenty-five to forty dollars per month.

Let us take a moment to define our terms. These are specialized definitions developed by the author. Piracy is

the act of copying published software with the intent of circumventing normal payment to publishers, authors, etc. Software security is the relative safety of software from piracy. Protection refers to the several processes which render programs difficult to transfer intact to another magnetic medium.

Early programs were supplied on cassette tape. Every APPLE computer sold had the ability to read and write to tape. It was very easy to LOAD a tape and then SAVE the BASIC or machine language program to another tape. Users did exactly that. They exchanged copies of tapes with the goal of minimizing costs. This process not only continued, but expanded. The inclusion of the floppy disk as a part of the APPLE II system made it possible to save programs more efficiently.

It is difficult, however, to collect a very large library of pirated programs on cassette tape. Such a library would be both difficult to manage in terms of size and would be expensive. A blank cassette tape cost between one and three dollars, and although the cost of one hundred tapes was far less than the cost of one hundred programs, it still represented a sizable sum.

The floppy disk, however, made storing a dozen programs on one five dollar diskette easy and efficient. A program can be loaded by cassette and saved onto the disk. Diskettes themselves can easily be copied. From the beginning Apple supplied a copy program with every disk system

they sold. The result was that collecting large libraries was not only feasable, but easy and inexpensive.

It is inevitable that such program exchanging would become organized. "Softtalk" magazine reports of computer clubs existing for the sole purpose of exchanging pirate copies of software. The author has personally observed this in Washington D.C. and Washington state user's groups. The Washington D.C. user's group went so far as to have an event which they called field day. The purpose of this event was to break the protection on the games currently marketed and it is naive to assume that once the protection was broken copies were not exchanged.

The obvious question at this point is what is the statistical extent of this practice and what is the overall effect on the software industry's economy. Softtalk magazine recently published an entire issue dedicated to software piracy.[YUEPT] In it, they quote Progressive Software President Neil Lipson as saying Progressive Software would have closed as a business due to software piracy, except for a hardware product which was a success last year.

On the subject of economic impact, they quote a survey by them in which they discovered that the average APPLE II owner possessed over a hundred dollars worth of pirated software. "Compound that figure by ten thousand new APPLE owners each month you arrive at a sum of one million dollars being siphoned out of the software industry monthly, and that counts only software designed for APPLES."[YUEPT]

"The flow-through theory of money postulates that each dollar spent within a relatively closed community, such as the software industry represents, will actually infuse that communities economy with $2.50 in the purchase of goods and services before it is exhaused. Using that theory, the loss to the software industry from piracy is actually closer to 2.5 million per month--1 million in actual loss and 1.5 million in lost opportunity." [YUEPT]

It is necessary to add an additional note to these figures. The above loss estimates are based on a minimum pirated library of $100. It has been the author's observation that the loss figure is frequently two to three times that reported in Softtalk, and the author has personally observed large libraries which were almost totally pirated material.

Softtalk reports that Progressive Software's President believes that retail dealers are the major sources of Pirated software. He gives as an example the "dealers who never fail to order one copy of a program but no more." "Jim Collins of OB2, a large computer and software distributer in Newport Beach California has been able to trace down roughly five thousand dollars worth of this type of illegimate software." They additionally report that one publisher gives the estimate that every copy he sells is copied two to three times [YUEPT]

This author has observed an APPLE computer dealer's salesman who systematically copied every program possible

which entered his store. These programs were made available free to friends and customers. Demonstration copies of programs sent to the retail store frequently never arrived. They were held at the parent company by an employee, the protection broken, and the resulting unprotected programs copied and given to at least one friend. From this point the copy was itself copied and distributed to a number of other individuals.

The author has been approached by the dealer's salesman and asked to unprotect copy protected software on two occassions. One was a popular game program, and the other was a demonstration program of an important business package. The purpose of the request was to supply this software to friends of the salesman.

This demonstrates the severity of the situation. The amount lost has led to software distributers attempting to copyprotect their software. This has led to a protection technology and a counter technology on the part of users.

Recently, a development occured which has shaken the very foundation of the industry. A program was developed and marketed by a firm which would copy protected software disks. This was a result of many firms copy protecting their software and not supplying back up copies or replacement copies if the original became damaged. As an example, Personal software in their VISICALC guarantee stated they would replace a disk which became defective in the first year for thirty dollars. They made no statement concerning

replacements after that period. The disk in question was a business program and sold for one hundred and fifty dollars. It is natural to assume than any business could ill afford any loss of program use, and the implied possibility of being forced to repurchase the entire package would not be tolerated. This in fact was the case. An individual developed a program to copy this specific disk. This was to be overshadowed by what was to come.

What was to come was a copy program which would copy any disk no matter what the protected format. It copied equally well VISICALC and Bill Budges Space Album. The software industry reacted with rage and alarm. In numerous conversations with software protection experts Steve Gibson of California Pacific and John Arkley of APPLE Computer, this was the primary topic of concern. [GIBPH] [ARKPH]

The program in question is named LOCKSMITH. The author purchased a copy. It is significant to note that the serial number was over four thousand and the product has only been available for four months. In addition most of the major microcomputer magazines refuse to advertise LOCKSMITH. [EMMPH] [GOLED]

In an experiment the author attempted to copy several major pieces of software with LOCKSMITH. It copied all but one without difficulty. The one used a system of placing the tracks on the disk at one half their normal spacing. An aternate keyboard command ordered Locksmith to check for this technique and adjust for it. The result was that this

disk also copied properly. The author's conclusions are that Locksmith is already a major tool for the software pirate and that no software is currently safe from piracy.

The remainder of this paper examines techniques to copy protect software. The author develops a set of techniques to provide a copy protected DOS. In addition, the author produces a disk format which LOCKSMITH cannot read and hence copy properly.

CHAPTER IV

COPY PROTECTION TECHNIQUES

The techniques required to protect software from piracy are responses to the piracy techniques themselves. There are two fundamental techniques of software piracy. The most common technique is to boot and run the disk, and then transfer control of the processor to the monitor. It is now possible to save the RAM image to cassette tape. This RAM image is the program and may be transferred to another disk or executed from the cassette. The second most common technique is to directly remove the program from the diskette, and copy it onto another disk.

From the above, hueristics indicate that two things must be done to prevent program piracy of a disk. The disk itself must be altered in such a manner as to be incompatible with standard DOS. This results in I/O errors if the software pirate attempts to catalog, copy, or otherwise remove any information from the disk. The second thing which must be done is to produce the program in such a way that the RAM image itself is not executable once control is returned to the monitor via the RESET function. Examination of numerous protected software packages indicates that these two points are indeed basic to all protection systems. The remainder of this paper examines methods to accomplish these tasks.

In order to discuss what is "standard DOS" and methods

required to make it incompatible, we must examine what is actually placed on a disk track. It must remembered that the disk head removes bits in a serial pattern, and when an entire byte has been read, the byte is transferred to the disk drives I/O card. This I/O card appears to contain an address whose contents change rapidly.

Data is not stored on the disk in a straight forward manner. APPLE computer uses a system of data transfer in which a byte is encoded as five or six bit "nibbles". Thirteen sector APPLE DOS uses five bit nibbles, while the sixteen sector DOS uses six bit nibbles. These nibbles are converted to bytes by indexing a nibble table. APPLE also refers to the resultant bytes as nibbles. In the thirteen sector DOS there are 32 valid nibbles, while in the sixteen sector DOS there are 62 valid nibbles. In order for a nibble to be valid, there are certain rules governing the allowable sequences of ones and zeros. Thirteen sector DOS may not have three zeros in a row, while sixteen sector DOS may not have more than one set of two zeros. In addition, certain valid nibbles are excluded. These nibbles are used as sector and data headers. The nibbles are the bytes "D5" and "AA" in hex. Omitting them from the nibble tables insures that they will not appear anywhere in data which might cause false attempts at sector alignment and a subsequently higher error rate.

DOS searches for a specific pattern to find the beginning of a sector. Following this is the volume, address,

and checksum information. Finally, the sector header is terminated by several bytes which serve as additional checks for proper alignment.

The volume, address, and checksum information is stored in sets of two bytes. It is decoded by storing the first byte, reading the second byte, rotating it right once, and performing an AND operation with the first byte. Thus, the entire sector and data header of track $00 sector $00 looks like this:

D5 AA B5 FF FE AA AA AA FE FE DE AA EB FF FF FF FF FF
FF FF FF FF FF FF FF FF FF D5 AA AD

Data follow this for four hundred and ten bytes. This is a thirteen sector format. There are minor differences in this and a sixteen sector format. The following is a standard sector and data header in sixteen sector format.

D5 AA 96 FF FE AA AA AA FE FE DE AA FF FF FF FF FF FF
FF FF FF D5 AA AD

The D5 AA 96 is termed a sector mark. The D5 AA AD is termed the data mark. Once again, data follow the data mark for but for only three hundred and thirty six bytes. The several bytes following the D5 AA 96 and the address data is a variable amount of space to allow overlap befor the data mark D5 AA AD begins.

Changing this format is enough to produce a disk which cannot be copied with standard copy programs, catalogued, or accessed with standard DOS in any manner. It is perfectly capable of being copied by Locksmith and other bit copiers,

however. These programs do not copy a sector at a time. Thus, they avoid the I/O errors which would be generated by attempting to read non standard formats.

A DOS which is able to read nonstandard format is in itself not sufficient to prevent software pirates from removing the program itself from RAM. MUSE software in Baltimore Maryland uses a DOS to protect all of their software. However, in most cases it preserves all of the standard commands, and more importantly can be commanded from the keyboard. As a result, even the most inexperienced software pirate can catalog the disk and load all programs on the disk one at a time into memory where they can subsequently be saved to cassette tape.

Keeping the above in mind, two additional requirements become imposed on an adequate protected DOS. First, the DOS must be rendered incapable of being operated from the keyboard. Furthermore, attempted keyboard operation should yield catastrophic results. Second, the RWTS must be so constructed so as to be inoperative unless used within the framework of the entire DOS. An alternate solution to the second condition would be to render the RWTS forever inoperative after a RESET condition occurs. The former solution is necessary in business applications where the format changes from protected to standard to handle data files. The latter solution is ideal for games where a single format will suffice. In addition, it is advisable to determine if the command to be executed orignates from the keyboard, or

from within a program. Direct commands from the keyboard are detected and result in core being reset.

The preceeding protects a disk from normal access and makes use of the DOS difficult. Pressing the RESET key, however disabls DOS unless an autostart ROM is used. The autostart ROM causes program execution to vector to the address located in locations $3F2 and $3F3 provided the value at location $3F4 is an exclusive OR of the value at $3F3 and $A5. Upon start up the XOR test fails and the monitor performs all initial start up operations including attempting to boot the disk system if present

These reset locations can branch to the RUN address, error handling routines or what ever else is appropriate. Without the autostart ROM, however, control is passed to the monitor. From the monitor, it is easy to return to BASIC with the program intact. The user may then SAVE a basic program to cassette tape.

Machine language programs can receive a measure of protection by imbedding the start point within the machine language code, rather than locating the start point at the beginning of the code. The DOS or other loader can also set several page zero locations to be destroyed upon reset. Graphics programs may also use the normal screen memory locations from $400 to $7FF, as these are also destroyed upon reset.

CHAPTER V

DEVELOPMENT OF PROTECTION TECHNIQUES

What follows is a review of the development of the protection techniques discussed in chapter IV. Weakness in these techniques is discussed together with solutions which increase the overall security.

The first protection technique discovered by the author simply involved moving the catalog from track $11 to track $10. Disks modified in such a manner are unusable in DOS 3.1-3.21. The catalog stands as a cross roads in any operating system. In it not only exists the names of all associated files, but the file type, certain protection data, and most importantly the track sector list which contains the track and sector address of each file block. Thus, to load a file it is necessary to find the catalog, for it acts as a gateway to the remainder of the diskette. Move the catalog to another track and it is hidden. Only a careful sector by sector search in combination with a byte for byte interpretation of the data therein could find it, or a disassembly of the protected operating system and understanding of the methods used in catalog access.

Copy programs frequently access the catalog and copy file by file. Sector zero of the catalog contains a master list of all sectors in use on the disk. This list is termed VTOC(volume table of contents). VTOC was necessary for all DOS 3.1-3.21 copy programs. Thirteen sector DOS did not

write a data mark (D5 AA AD) following a sector mark (D5 AA B5) unless data was actually stored in the sector. Attempts to read the sector, would thus fail. A copy program would generate numerous I/O errors if it attempted to read unused sectors. Therefor, all thirteen sector copy programs read VTOC on track $11 to determine which sectors to read. If track $11 was unformated, or if track $11 sector 0 had not been written to, the copy program would generate an I/O error and stop. The reader should be aware of a psychological factor concerning I/O errors. When an I/O error is encountered, the DOS transfers execution to a track seek routine which recalibrates the read/write head by attempting to reset it several times to track zero. This attempt at resetting causes the head to hit a stop repeatedly. This causes a loud rattling noise which most users find intensely disturbing.

The technique of track switching was quickly determined to be unsatisfactory. A telephone call to the Apple Pugeot Sound Library Exchange revealed this technique to be common knowledge. Later issues of the this user's group news magazine contained information required to reconvert this to a standard format. The author, himself, discovered this technique while unprotecting a disk restructured in this manner. The technique is unusable entirely with sixteen sector DOS. The sixteen sector format always writes a data header, and copy programs copy a track at a time without consulting VTOC. Track $11 would have to be unformated entirely, and

any user with patience could let his copy program generate I/O errors for track $11. Subsequent copies of the copy would not generate I/O errors at all.

The technique attempted next involved modifying the sector. The author did not reference page zero in the beginning. The disk initialized in this manner did initialize properly, it was possible to access the catalog, and otherwise appeared to operarte properly. The initializion process itself does three things. The most fundimental aim of initializion is to format the diskette into tracks and sectors which can be addressed. DOS also copies itself onto tracks zero through two of the diskette in a manner which will boot. In order to make turnkey systems possible, DOS also copies the BASIC program in memory onto the newly initialized diskette. The first of these processes were tested. A standard DOS 3.3 was then loaded. As expected, the newly initialized disk would not read and was impossible to access as a result of the format change. Next, an attempt was made to reboot the new disk. It would not boot. The difficulty became quickly evident. The booting process used a program in ROM to read track zero, sectors zero through nine. These must be in standard format, or the disk will not boot. Converting the format turned out to be easy. The Apple Puget Sound Library Exchange produced a program which they termed "Disk Zap". This program allows reading and writing to individual sectors of a disk. By changing the data read routine to conform to the altered format it was

possible to read the sector. By leaving the write format intact and rewriting the data to the sector, the data header was itself rewritten in normal format. Sectors zero through nine were rewritten in this manner. The result was that the disk then booted and appeared to operate normally.

Writing a copy program which would automatically switch formats for both reading and writing proved difficult at first. APPLE'S standard copy program was available to modify. The difficulty was encountered in disassembling and understanding the machine language portion of the program. It was necessary to determine which track and sector was being accessed, and then modify the RWTS itself. The problem was solved by vectoring to a subroutine just prior to calling RWTS. RWTS receives the track and sector information as well as other parameters via a table which APPLE terms the IOB. This is located at $B7E8 in a 48K APPLE. Track and sector data were accessed from this table, and the correct bytes required to read and write the data header were stored directly in RWTS replacing the format information already there. The copy program would not work in a computer with less memory since RWTS would no longer reside in the same location.

In order to make the DOS unusable, the LOAD, SAVE, and RUN commands were disabled by replacing their listing in the command table with characters which could not be input from the keyboard. The DOS itself could be executed and commands used from the keyboard, but once denied the ability to cata-

log the protected disk, a user could not determine the name of files he wished to load.

This system was far from satisfactory. It was memory size dependent from the aspect of originating copies.

From a user's viewpoint there was an additional problem. The disk only operated in a protected format. It is necessary to provide a DOS which produces unprotected text files. An additional requirement was voiced by a programmer in Greenville, S.C. His requirements included the ability of DOS to write to the protected diskette. At the time, this proved to be difficult to overcome. The inadvisability of directly modifying the DOS itself had become evident. The technique of referencing page zero had been established. The attempt to use the same technique while writing proved to be unworkable due to timing problems. Thus, it appeared to be possible to write only in one format.

Steve Gibson of California Pacific offered the suggestion of modifying other parts of the format. Writing could still be in a single unprotected format, but writing can only occure if the desired sector can be located. This appeared to solve the immediate problem, but created another one in its place. The booting technique requires that track zero, sectors zero through nine be readable by the software in the boot ROM. RWTS, however, must read the protected format in order that the DOS itself be protected. This created a problem, since the booting process begins on track two and ends on track zero. To continue this booting prac-

tice, track zero must be written in two formats. This was beyond the ability of the author, and John Arkley, APPLE's protection expert, reported that to his knowledge that problem had never been solved. The problem was avoided, however by rearranging the sectors on the first three tracks. The original list began on track two sector four and ended on track zero sector ten. By shifting the this six sectors, the final booting stage began on track two sector ten and ended on track one sector zero.

It was not difficult to initialize an entire disk in protected format, modify the test which limited the number of tracks to initialize and reinitialize track zero in an unprotected format.

One aspect of this protected DOS has not been discussed. The initializion process is necessary in many business programs to create data disks, and yet at the same time this would serve to unprotect the program by storing it on the data diskette. In addition, a copy of the DOS would also be saved in unprotected format. Since this was not acceptable a method had to be found to circumvent it. John Arkley's solution to the problem was in three parts. First location $A397 in DOS was changed to $A2EA. This closed the BASIC file before any writing could be accomplished. The jump to the subroutine which deposits DOS itself to the disk was changed to reference a return from subroutine statement elsewhere in the DOS. Finally, he suggest changing a mask byte which freed up tracks one and two. The suggestions

worked properly and were implemented, thus solving this aspect of protection.

It appears now, the major weakness of the system is two fold. First, it was possible to access track zero with a program such as disk zap The second threat involved the program Locksmith which copied altered formats with impunity.

The discussion of Locksmith and the technique required to neutralize its threat will follow in chapter VI. The other threat was easily and quickly solved. It is possible to alter the format in such a way as to make it unaccessable to sector reading programs and still boot properly.

CHAPTER VI

TECHNIQUES TO DEFEAT NIBBLE COPIERS

The threat of Locksmith to the programming industry cannot be underestimated. A similar program is being distributed by Sensible Software called "Back It Up". Both programs read disks a byte at a time, regardless of format, and write the contents of the tracks to another disk. These programs are termed nibble copiers or bit copiers. Several individuals have developed bit copiers of their own, but do not market them.

Attempts to produce disks which could not be copied by bit copiers most frequently rely on locating tracks one half track away from their normal location.

Both commercial bit copiers were obtained by the author and studied. The theory behind any bit copier involves an attempt to copy a track without damaging the data. Devices to copy cassettes regardless of format have existed for years. They are simple in concept: read a bit and at the same time, write the bit to another recorder. It is not as simple in the case of a rapidly moving disk. Each track of a disk is circular in nature. Thus, it is very difficult to determine whether the head has read the entire track or has read additional bytes and thus provided overlap. An APPLE disk drive will store approximately eighteen thousand ninty five characters per track. In simplified theory you can read these bytes and write them to the copy diskette. This theory does not, however, take into account minute differ-

ences in speed between the drive which produced the disk and the drive which uses it, nor the speed differences which occur in the same drive. APPLE computer has a set of diagnostics for repair of their computers and disk drives. It includes a speed adjustment program. They measure the speed in arbitrary units. The instructions say a drive is properly adjusted if it is plus or minus 26 Disk speed units. In practice, it is not possible to achieve a stability greater than plus and minus three D speed units. This amounts to plus and minus several bytes in the case of attempting to read and write a track. If these bytes occur between sectors, there is no harm done. If, however, the pattern of the sector or data itself is disturbed an I/O error will occur. The checksum will no longer match if the track is over written inexactly. In practice these errors destroy one sector per track. Thus, a disk will fail to boot since its data crosses several tracks, and programs will fail to load.

Nibble copiers are able to re-write a track without destroying either the sector or the data it contains. Most bit copiers attempt error checking of one form or another. One form of error checking attempts to find an approximation of a sector.

The author experimented with several methods of defeating bit copiers. A format can be produced which will confuse the alignment process of a bit copier. The author initialized a disk which altered its format in such a way as to be unreadable by all current nibble copiers and still func-

tion properly. LOCKSMITH generated error message "4" for each track. This is the cannot read reliably message. The copy diskette would not boot. It is significant to note that the speed of the originating disk and the copy disk drive had previously been matched as close as the APPLE Disk speed test would allow. Sensible Software's Back It Up reacted to the new format even more severly. Back It Up never left the read mode in the first track. The copy process was stoped completely and the contents of the track were never analyzed. The closeness of Locksmith's copy would be an optimum response, and further speed differences would degrade the copying even further. Such speed differences between drives is a reasonable assumption.

CHAPTER VII

CONCLUSIONS

Like cryptography, copy protecting software is an ever escalating technology. If it is indeed possible to render bit copiers useless, it is still possible to read a track from a protected diskette into memory and visually establsh the format. Modifying RWTS to read the new format, although impossible for most users, still will be acomplished by someone. Once unprotected, the copies will be traded or sold and the software industry is no better off than it was before the bit copiers became available. Users can still resort to more sophisticated methods for removing software intact from RAM. One such method has already been accomplished. program overlays would deter this practice to some unknown degree, but in the end patience would result in the user obtaining the entire program.

A few protection techniques bear mentioning. All can be defeated with varying degrees of users expertise. One protection concept requires a special hardware module for the program to perform properly. BPI Software utilizes a simple module which plugs into the game paddle socket. The programs are written in APPLESOFT BASIC, and finding the references to the module were trivial. However, to the non programmer such a device is preventing a great deal of pirating. The user has a system he can back up himself, and update from time to time. A similar concept requires a com-

plicated ROM be placed in one of the I/O sockets. This has potential if a subroutine in included in the ROM which will be difficult to relocate. It is only a matter of time, however, until a RAM board is marketed which will take the place of the ROM module. Pirates will then copy the ROM as they now copy the program and simply store a copy into the RAM board.

Legal aspects of software protection have not been discussed. Any conclusions, however, without such a mention are meaningless. The problem has been brought on by both the legal system and the software industry itself. Software piracy is not punished. Software pirates work with impunity. Police do not investigate this type of crime. Software distributers do not send out investigators. A software pirate has only his conscience and the opinion of his peers as a limiting factor. Here, as elsewhere, that appears to be insufficient.

It would appear that the best direction for future actions in this area would be in the non technical arena. Software piracy must be investigated, tried, and punished like any other crime. Software must be available at prices which by virture of being low would make the crime of software piracy not worth the few additional dollars required for an original copy of the magnetic medium together with a shiny and appealing manual.

BIBLIOGRAPHY

APPUT Untitled manuscript for software vendors APPLE Computer 1980

APPAS APPLESOFT BASIC programming manual APPLE Computer 1978

APPRM APPLE Reference Manual APPLE Computer 1979

APPDS APPLE Disk Operating System Manual APPLE Computer 1980

ARKPH Arkley, John Telephone conversations: Aug 1980, Oct 1980, Jan 1981, April 1981

BUDII Budge, Joseph H. Inside Initialization, CALL A.P.P.L.E. Vol IV, No. 2: Apple Pubget Sound Program Library Exchange

BURDB Burns, Ted Apple DOS Booting Process, The APPLE Orchard, Vol I, No. 1

EMMPH Emery, Allen telephone conversations: April 1980

GOLED Golding, Val Editorial Bit Copiers, CALL A.P.P.L.E. VOL IV, No.2

GIBPH Gibson, Steve telephone conversations: Oct 1980, Jan 1981, April 1981

HARPH Hartley, Charles telephone conversations: April 1980

LANUM Landsman, Richard Using Muffin With VTOC and Directory on $10, CALL A.P.P.L.E. Vol. IV, No. 2

PUMDI Pump, Mark DOS Internals: An Overview, CALL A.P.PL.E. Vol. IV No. 2

ROBPH Robinson, Allan telephone conversations: Jan 1981, March 1981, April 1981

YUEPT Yuen, Matthew Pirate Thief. Who Dares Catch Him?, Softtalk, Vol. 1 No. 2

ZAKP6 Zaks, Rodnay Programming the 6502, Published by SYBEX INC. 1978